```
1    /*
2     *  %W% %E% Everett Stoub
3     *
4     *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
5     *
6     *  ez_text applet displays snaking text with in-line graphics
7     *       and produces a reader screen frame when clicked
8     */
9
10   /**
11    *  @author Everett Stoub %I%, %G%
12    */
13
14   import java.awt.*;
15   import java.applet.*;
16   import java.net.*;
17   import java.io.*;
18   import java.util.*;
19
20   public class ez_text extends Applet
21   {
22       private final boolean formattedText     = true;    //  if true, in-line graphics
23   and pull quotes are enabled
24
25       private String           title          = null;
26       private String           article        = null;
27       private ezrpnlap         panel          = null;
28       private ezrfrmap         frame          = null;
29
30       private MediaTracker     tracker        = null;
31       private Vector           imageFileNames = null;    //  vector of all image file
32   names for document
33       private Vector           graphicsURLs   = null;    //  vector of all graphic
34   image URLs for document
35     .   private Vector           graphicsImages = null;    //  vector of all graphic
36   images for document
37
38       private final boolean    frameOption    = true;    //  set false for 1Kb code
39   reduction
40
41       private final int        NONE           = 0;
42       private final int        LINE           = 1;
43       private final int        FLAT           = 2;
44       private final int        RAISED         = 3;
45       private final int        DROPPED        = 4;
46       private final int        ETCHED         = 5;
47
48       private int              frameType      = NONE;
49       private int              frameSize      = 0;
50
51       private final String     AG = "`";
52       private final String     AM = "&";
53       private final String     SC = ";";
54       private final String     SP = " ";
55       private final String     PL = "<p";
56       private final String     QL = "\"";
57       private final String     PR = "<p>";
58       private final String     QT = "quot";
59
60       private final String     alert_0 = "PLEASE READ - A Java ";
61       private final String     alert_1 = "Exception has occurred. ";
62       private final String     alert_2 = "Most likely, there has been a transmission
63   failure with one or more applet files. A Refresh or Reload command on your browser
64   will sometimes lead to a successful download. ";
65       private final String     alert_3 = "An attempt to read a local file has been
66   detected! In order to read this article, you must download the applet using hyper-text
67   transport protocols (http://...) from a web server. Please try again using http with a
68   valid URL. ";
69       private final String     alert_4 = "If the problem persists, please contact the Web
70   Master.";
```

```
1
2      public final void dbg(String s)
3      {
4          if (debug) System.out.println(s);
5      }
6      private boolean debug = false;       //  Print debugging info?
7
8      public String getTitle()
9      {
10     //  dbg("ezrpnlap.getTitle()");
11
12         return this.title;
13     }
14
15     public final void paint(Graphics g)
16     {
17     //  dbg("ez_text.paint(): frameType ="+frameType);
18
19         if (frameOption)
20         {
21             int i;
22             Dimension d = size();
23             switch (frameType)
24             {
25                 case LINE:
26                 {   g.setColor(getForeground());
27                     for (i = 0; i < frameSize; i++)
28                         g.drawLine(0,d.height-i-1,d.width-i-1,d.height-i-1);
29                 }   break;
30                 case FLAT:
31                 {   g.setColor(getForeground());
32                     for (i = 0; i < frameSize; i++)
33                         g.drawRect(i,i,d.width-2*i-1,d.height-2*i-1);
34                 }   break;
35                 case RAISED:
36                 case DROPPED:
37                 case ETCHED:
38                 {   boolean raise = frameType==RAISED;
39                     int R = (getForeground().getRed()+  getBackground().getRed()   )/2;
40                     int G = (getForeground().getGreen()+getBackground().getGreen())/2;
41                     int B = (getForeground().getBlue()+ getBackground().getBlue()  )/2;
42                     g.setColor(new Color(R,G,B));
43                     for (i = 0; i < frameSize; i++)
44                     {
45                         if (frameType == ETCHED) raise = i >= frameSize/2;
46                         g.draw3DRect(i,i,d.width-2*i-1,d.height-2*i-1,raise);
47                     }
48                     if (frameType == ETCHED)
49                     {
50                         g.setColor(getForeground());
51                         g.drawLine(0,0,frameSize/2,frameSize/2);
52                         g.drawLine(d.width-frameSize,d.height-frameSize,d.width-
53     frameSize/2,d.height-frameSize/2);
54                         g.setColor(getBackground());
55                         g.drawLine(frameSize/2,frameSize/2,frameSize,frameSize);
56                         g.drawLine(d.width-frameSize/2,d.height-frameSize/2,d.width-
57     1,d.height-1);
58                     }
59                     else
60                     {
61
62     g.setColor((frameType==RAISED)?getBackground():getForeground());
63                         g.drawLine(0,0,frameSize,frameSize);
64
65     g.setColor((frameType==RAISED)?getForeground():getBackground());
66                         g.drawLine(d.width-frameSize,d.height-frameSize,d.width-
67     1,d.height-1);
68                     }
69                 }
70             }
```

```
1              }
2          }
3
4      public synchronized Image fetchImage(String fileName)
5      {
6      //  dbg("ez_text.fetchImage("+fileName+")");
7
8          if (!formattedText || fileName == null)
9          {
10             return null;
11         }
12         else
13         {
14             Image theImage = null;
15
16             int imageDatabaseIndex =                        //  get reference to image
17     database file name
18                 imageFileNames.indexOf(fileName);
19
20             if (imageDatabaseIndex < 0)                     //  not registered yet:
21     add it in
22             {
23                 URL theURL = null;
24
25                 try                                         //  look in the
26     documentBase
27                 {
28                     theURL = new URL(getDocumentBase(), fileName);
29                     InputStream is = theURL.openStream();   //  test connection
30                     theImage = getImage(theURL);
31                     tracker.addImage(theImage,0);
32
33                 //  jump-start image loading
34
35                     tracker.checkID(0,true);
36                     try   { tracker.waitForID(0);         }
37                     catch ( InterruptedException ie ) { }
38                     Image bufferedImage = createImage(1,1);
39                     if (bufferedImage != null)
40                     {
41                         Graphics bg = bufferedImage.getGraphics();
42                         if (bg != null)
43                         {
44                             bg.drawImage(theImage,0,0,this);
45                             bg.dispose();
46                         }
47                     }
48                     bufferedImage.flush();
49                 }
50                 catch (MalformedURLException u)
51                 {
52                     System.out.println("ez_text.fetchImage("+fileName+"):
53     MalformedURLException = "+u);
54                     theImage = null;
55                 }
56                 catch (IOException io)
57                 {
58                     System.out.println("ez_text.fetchImage("+fileName+"): InputStream
59     IOException = "+io);
60                     theImage = null;
61                 }
62
63             //  update image database for document
64
65                 imageFileNames.addElement(fileName);        //  the filename is the
66     main key to image database access
67                 graphicsURLs.addElement(theURL);            //  in case we need to
68     download it again later
69                 graphicsImages.addElement(theImage);        //  this may be reset to
70     null with imageUpdate
```

```
1                      }
2                  else
3                  {
4                      theImage = (Image)graphicsImages.elementAt(imageDatabaseIndex);
5                  }
6
7                  return theImage;
8              }
9          }
10
11      public void disposeFrame()
12      {
13      //  dbg("ez_text.disposeFrame()");
14
15          if (frame != null)
16          {
17              Event evt = new Event(frame, Event.WINDOW_DESTROY, null);
18              Component co[] = frame.getComponents();
19              for (int i = 0; i < co.length; i++)
20                  co[i].handleEvent(evt);
21
22              frame.dispose();
23              frame = null;
24          }
25      }
26
27      public boolean mouseUp(Event evt, int x, int y)
28      {
29      //  dbg("ez_text.mouseUp(Event "+evt+", int "+x+", int "+y+") ");
30
31          if (frame != null)
32              frame.toFront();
33          else
34              frame = new ezrfrmap(title, article, this);
35
36          return false;
37      }
38
39      public String htmlCanon(String itsText, boolean convertNWSs)
40      {
41      //  dbg("ezrpnlap.htmlCanon(\n--------------"+itsText+"\n--------------,
42  "+convertNWSs+")");
43
44          if (!formattedText)
45              return itsText;
46
47          if (convertNWSs)                                      //  look for
48  [`title<p]...
49          {
50              String LCtext = itsText.toLowerCase();
51
52              if (LCtext.startsWith(AG))                        //  initial accent
53  grave specifies article title
54              {
55                  int endTitle = LCtext.indexOf(PL);            //  end of line after
56  canonization
57                  if (endTitle > 0) ,
58                  {
59                      title = itsText.substring(1,endTitle);
60                      itsText = itsText.substring(endTitle);
61                  }
62              }
63          }
64          else
65          {
66          //  replace character entities with spaces (except quot -> \")
67
68              StringTokenizer entities = new StringTokenizer(itsText,AM,true);
69              StringBuffer newText = new StringBuffer(70);
70
```

```
1              while (entities.hasMoreTokens())
2              {
3                    String entity =                              //    [body
4     text],[&][#nnn;body text],...,[&][#nnn;body text]
5                          entities.nextToken(AM);
6
7                    if (entity.equals(AM))
8                    {
9                          String eText =                         //    should be
10    character entity text #nnn or name
11                                entities.nextToken(SC);
12
13                          entity = entities.nextToken();        //    should be ";" if
14    eText is valid character entity value
15
16                          if (entity.equals(SC))                //    replace character
17    entity with local character (set)
18                          {
19                                if (eText.equals(QT))
20                                {
21                                      newText.append(QL);
22                                }
23                                else
24                                {
25                                      newText.append(SP);
26                                }
27                          }
28                          else                                  //    syntax error:
29                          {
30                                newText.append(AM).append(eText).append(entity);
31                          }
32                    }
33                    else
34                    {
35                          newText.append(entity);               //    standard text
36                    }
37              }
38              itsText = newText.toString();
39
40        //    look for [<title...>title text</title...]
41
42        String LCtext = itsText.toLowerCase();
43
44        int ending = 0, starts = LCtext.indexOf("<title");
45        if (starts > -1)
46        {
47              starts = LCtext.indexOf(">",starts)+1;            //    start of title
48              if (starts > 0 && starts < LCtext.length())
49              {
50                    ending = LCtext.indexOf("</title",starts);
51                    if (ending > -1)
52                    {
53                          title = itsText.substring(starts,ending);
54                    }
55              }
56        }
57
58        //    look for [<body...>body text</body...]
59
60        starts = LCtext.indexOf("<body");
61        if (starts > -1)
62        {
63              starts = LCtext.indexOf(">",starts)+1;            //    start of body
64              if (starts > 0 && starts < LCtext.length())
65              {
66                    ending = LCtext.indexOf("</body",starts);
67                    if (ending > -1)
68                    {
69                          return itsText.substring(starts,ending);
70                    }
```

```
 1                         }
 2                      }
 3                  }
 4              return itsText;
 5          }
 6
 7      public String lineCanon(String itsText, boolean convertCRs)
 8          {
 9      //  dbg("ez_text.lineCanon()");
10
11          if (itsText == null
12          ||  itsText.length() == 0)   return itsText;
13
14          String newSeparator = convertCRs?PR:SP;                 //  target return code
15          String searchSeparators[] =                            //  replacable return
16  codes
17              {
18                  "\n"+"\r",                                     //  chr(10), chr(13)
19                  "\r"+"\n",                                     //  chr(13), chr(10)
20
21                  "\r",                                          //  chr(13)
22                  "\n",                                          //  chr(10)
23                  "\f"                                           //  chr(12)
24              };
25          String newText = new String(itsText);
26
27          for (int i = 0; i < searchSeparators.length; i++)      //  replace old with
28  new
29              {
30                  String oldSeparator = new String(searchSeparators[i]);
31                  newText = replaceString(newText,oldSeparator,newSeparator);
32              }
33          return newText;
34          }
35
36      public String replaceString
37          (   String theString                                  //  original string
38          ,   String oldString                                  //  replace-ee
39          ,   String newString                                  //  replace-or
40          )
41          {
42      //  dbg("ez_Text.replaceString(theString, old: "+oldString+", new:
43  "+newString+")");
44
45          StringBuffer out = new StringBuffer();
46
47          int oldIndex = -1, oldEnd = 0;
48          int oldLength = oldString.length();
49
50          while ((oldIndex = theString.indexOf(oldString, oldEnd)) > -1)
51              {
52                  String piece = theString.substring(oldEnd,oldIndex);//  got it? get it!
53  good...
54                  out.append(piece);                             //  add the piece
55  before the next old string
56                  out.append(newString);                         //  get the
57  replacement
58                  oldEnd = oldIndex + oldLength;                  //  skip to the end of
59  this old string
60              }                                                  //  nothing more to
61  replace
62          if (oldEnd < theString.length())
63              {
64                  String piece = theString.substring(oldEnd);
65                  out.append(piece);                             //  add the piece
66  after the last old string
67              }
68          return out.toString();
69          }
70
```

```
1          public Insets insets()
2          {
3          //   dbg("ezrpnlap.insets()");
4
5              if (frameOption) switch (frameType)
6              {
7                  case LINE:
8                      return new Insets(0,0,frameSize,0);
9                  case FLAT:
10                 case RAISED:
11                 case DROPPED:
12                 case ETCHED:
13                     return new Insets(frameSize,frameSize,frameSize,frameSize);
14             }
15             return new Insets(0,0,0,0);
16         }
17
18         public void init()
19         {
20         //   dbg("ez_text.init<>: parent background = "+getParent().getBackground());
21
22             if (formattedText)
23             {
24                 imageFileNames  = new Vector(10);                        //  must be created
25 before calls to fetchImage
26                 graphicsURLs    = new Vector(10);                        //  must be created
27 before calls to fetchImage
28                 graphicsImages  = new Vector(10);                        //  must be created
29 before calls to fetchImage
30             }
31
32             tracker = new MediaTracker(this);
33
34             String s = getParameter("fileName");
35             if (s != null) try                                          //  new URL, etc.
36             {
37                 URL url = new URL(getDocumentBase(), s);
38                 InputStream is = url.openStream();
39                 BufferedInputStream bis = new BufferedInputStream(is);
40
41                 int len = 0;
42                 byte buffer[] = new byte[512];
43                 StringBuffer pageText = new StringBuffer();
44                 while ((len = bis.read(buffer)) > 0)
45                     pageText.append(new String(buffer,0,0,len));
46
47                 bis.close();
48                 is.close();
49
50                 boolean toHTML = s.toLowerCase().endsWith(".txt") ||
51 s.toLowerCase().endsWith(".text") || s.toLowerCase().endsWith(".nws");
52                 article = htmlCanon                                      //  retains only
53 supported tags
54                     (   lineCanon(pageText.toString()
55                     ,   toHTML)                                          //  true converts
56 'CR's to '<p>'s
57                     ,   toHTML                                           //  true converts
58 enews tags to html
59                     );                                                   //  strips off head &
60 saves title
61
62             //  start loading in-line graphics right away
63
64                 StringTokenizer images = new StringTokenizer(article,"@",true);
65                 while (images.hasMoreTokens())
66                 {
67                     String token = images.nextToken();
68                     if (token.equals("@"))
69                     {
```

```
1                        String fileName = images.nextToken("<");    //   <p> following
2    fileName
3                        if (fileName.toLowerCase().endsWith(".jpeg")
4                        || fileName.toLowerCase().endsWith(".gif"))
5                            fetchImage(fileName.trim());
6                    }
7                }
8            }
9        catch (MalformedURLException mu)
10       {
11            StringBuffer msg = new StringBuffer();
12            msg.append(alert_0).append("Malformed URL
13   ").append(alert_1).append(alert_2).append(alert_4);
14
15            article = msg.toString();
16
17            String e = "Could not open "+s;
18            showStatus(e);
19            System.out.println(e+"; exception: "+mu);
20       }
21       catch (IOException io)
22       {
23            StringBuffer msg = new StringBuffer();
24            msg.append(alert_0).append("Input/Output
25   ").append(alert_1).append(alert_2).append(alert_4);
26
27            article = msg.toString();
28
29            String e = "Could not open "+s;
30            showStatus(e);
31            System.out.println(e+"; exception: "+io);
32       }
33       catch (SecurityException se)
34       {
35            StringBuffer msg = new StringBuffer();
36            msg.append(alert_0).append("Security
37   ").append(alert_1).append(alert_3).append(alert_4);
38
39            article = msg.toString();
40
41            String e = "Could not open "+s;
42            showStatus(e);
43            System.out.println(e+"; exception: "+se);
44       }
45       else
46       {
47            StringBuffer msg = new StringBuffer();
48            msg.append(alert_0).append("Input/Output ").append(alert_1);
49            msg.append("No article file specification was detected! ");
50            msg.append(alert_2).append(alert_4);
51
52            article = msg.toString();
53
54            String e = "No fileName specified";
55            showStatus(e);
56            System.out.println(e);
57       }
58       this.setLayout(new BorderLayout());
59       this.setBackground(getParent().getBackground());
60       this.setForeground(getParent().getForeground());
61
62       panel = new ezrpnlap(article, this, false);                        //  NOT the reader
63   screen
64
65       //  add insets for applet boundaries...
66       this.add("Center", panel);
67
68       if (frameOption)
69       {
70            String b = getParameter("frameType");
```

```
1                    if (b != null)
2                    {
3                        b = b.toUpperCase();
4                        if (b.equals("LINE"))    {   frameSize = 1;   frameType = LINE;    }
5                        if (b.equals("FLAT"))    {   frameSize = 1;   frameType = FLAT;    }
6                        if (b.equals("RAISED")) {    frameSize = 2;   frameType = RAISED; }
7                        if (b.equals("DROPPED")){    frameSize = 2;   frameType = DROPPED;}
8                        if (b.equals("ETCHED")){     frameSize = 4;   frameType = ETCHED; }
9                    }
10
11                   String w = getParameter("frameSize");
12                   if (w != null)
13                   try    {frameSize = Integer.parseInt(w); }
14                   catch (NumberFormatException nf)      {   }
15               }
16           }
17   }
18
19   /*
20    *    %W% %E% Everett Stoub
21    *
22    *  Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
23    *
24    *  ez_text applet reader screen frame
25    */
26
27   /**
28    *  @author Everett Stoub %I%, %G%
29    */
30
31   import java.awt.*;
32   import java.applet.*;
33   import java.net.*;
34   import java.io.*;
35   import java.util.*;
36
37   public final class ezrfrmap extends Frame
38   {
39       private ezrpnlap     readerScreen;
40       private Panel        control;
41       private ez_text      ez_Text;
42   //  private Label        footer;
43       private boolean      demo = true;
44
45       private String buttonName[] =
46       {    "Home"
47       ,    "Previous"
48       ,    "Next"
49       ,    "Font +"
50       ,    "Font -"
51       };
52
53       public ezrfrmap(String title, String readerContent, ez_text ez_Text)
54       {
55           super(title);
56
57   //  dbg("ezrfrmap.<init>");
58
59           this.ez_Text = ez_Text;
60           .this.readerScreen = new ezrpnlap(readerContent, ez_Text, true);
61           this.add("Center", readerScreen); .
62
63           control = new Panel();
64           control.setBackground(Color.gray);
65   //      control.setLayout(new FlowLayout(FlowLayout.RIGHT));
66           control.setLayout(new FlowLayout(FlowLayout.CENTER));
67
68           StringBuffer version = new StringBuffer();
69           version.append("ION EZ Text ");
70           if (demo)
```

```
1                         version.append("Demo");
2                 else
3                     version.append("v1.03");      // [Δn-major].[Δn-minor][Δn-tweak][Δa-fix]
4
5             control.add(new Label(version.toString()));
6             for (int i = 0; i < buttonName.length; i++)
7             {
8                 Button b = new Button(buttonName[i]);
9                 b.setBackground(Color.white);
10                control.add(b);
11            }
12            control.add(new Label("(c)'97 ION Systems, Inc. 888.ION.JAVA"));
13
14            this.add("South", control);
15
16            this.pack();
17            this.show();
18        }
19
20        public boolean action(Event e, Object o)
21        {
22            if (e.target instanceof Button)
23            {
24                if (o == (buttonName[0]))    readerScreen.doViewerHome();          else
25                if (o == (buttonName[1]))    readerScreen.doViewerLeft();          else
26                if (o == (buttonName[2]))    readerScreen.doViewerRight();         else
27                if (o == (buttonName[3]))    readerScreen.doViewerSize(false);     else
28                if (o == (buttonName[4]))    readerScreen.doViewerSize(true);      else
29    return false;
30            //  repaint();
31                return true;
32            }
33            else return false;
34        }
35
36    /*   private final void dbg(String  s)
37         {
38              if (debug) System.out.println(s);
39         }
40        private boolean debug = false;       //  Print debugging info?
41    */
42        public final boolean keyDown(Event evt, int key)     //  note:  keyUp is not
43    utilized at all (missing in JDK 1.02 on Mac)
44        {
45        //  dbg("ezrfrmap.keyDown("+evt+", "+key+")");
46
47            boolean down = key == Event.DOWN;
48
49            switch (key)
50            {
51            case Event.HOME:        readerScreen.doViewerHome();            repaint();   return
52    true;
53            case Event.PGDN:
54            case Event.RIGHT:       readerScreen.doViewerRight();           repaint();   return
55    true;
56            case Event.PGUP:
57            case Event.LEFT:        readerScreen.doViewerLeft();            repaint();   return
58    true;
59            case Event.UP:
60            case Event.DOWN:        readerScreen.doViewerSize(down);        repaint();   return
61    true;
62            }
63
64            return false;
65        }
66
67        public final boolean handleEvent(Event evt)
68        {
69        //  dbg("ezrfrmap.handleEvent("+evt+")");
70
```

```
1              switch (evt.id)
2              {
3              case Event.WINDOW_EXPOSE:      readerScreen.requestFocus();    break;
4              case Event.WINDOW_DEICONIFY:   this.show();                    break;
5              case Event.WINDOW_ICONIFY:     this.hide();                    break;
6              case Event.WINDOW_DESTROY:     ez_Text.disposeFrame();         break;
7              }
8              return super.handleEvent(evt);
9          }
10
11     public void update(Graphics g)
12         {
13     //    g.setColor(getBackground());
14     //    g.fillRect(0, 0, width, height);
15           g.setColor(getForeground());
16           paint(g);
17         }
18  }
19
20  /*
21   *   %W% %E% Everett Stoub
22   *
23   *   Copyright (c) '96-'97 ION Web Products, Inc. All Rights Reserved.
24   *
25   *   web crawler with ReaderScreen
26   */
27
28  /**
29   *   @author Everett Stoub %I%, %G%
30   */
31
32  import java.awt.*;
33  import java.awt.image.*;
34  import java.applet.*;
35  import java.net.*;
36  import java.io.*;
37  import java.util.*;
38
39  public final class ezrpnlap extends Panel
40  {
41      public final static int     LEFT      = 0;
42      public final static int     CENTER    = 1;
43      public final static int     RIGHT     = 2;
44      public final static String  MT        = "";
45      public final static String  BL        = " ";
46
47      private boolean     demo                    = true; //   mild display spam
48      private final boolean formattedText         = true; //   if true, in-line graphics and
49  pull quotes are enabled
50
51      private ez_text     ez_Text             = null;
52
53      private String      readerContent       = null;
54      private Image       readerScreen        = null;
55
56      private int         readerFontSize[]    = { 8,
57  9,10,11,12,14,18,24,30,36,42,48,56,64,72,80,96};
58      private int         readerFootIndex     = 2;    //   this 2-step decrement is also
59  hard coded in stepFontSize()
60      private int         readerSizeIndex     = 4;    //   default for on-webPage article
61  rendering
62      private int         readerPullIndex     = 5;    //   this 1-step increment is also
63  hard coded in stepFontSize()
64      private int         readerHeadIndex     = 6;    //   this 2-step increment is also
65  hard coded in stepFontSize()
66      private Font        readerFooterFont    = null;
67      private Font        readerScreenFont    = null;
68      private Font        readerPQuoteFont    = null;
69      private Font        readerHeaderFont    = null;
70
```

```
1        private int            readerAlignment     = LEFT;
2        private int            readerPerimeter     = 10;
3        private int            readerNumCols       = 1;
4        private int            readerLeading       = 0;
5        private int            readerPageNum       = 1;
6        private int            readerNumPages      = 0;
7        private int            readerEstPages      = 1;
8        private int            readerEndIndex      = 0;
9        private int            readerStartIndex    = 0;
10       private int            readerTextLength    = 0;
11       private boolean        readerScreenDirty   = true;      // true if panel needs redraw
12       private boolean        readerFrame         = false;     // true if panel owned by
13   frame
14       private boolean        readerHeader        = false;     // true if article begins with
15   `title
16
17       private Color          readerBackColor     = new Color(231,231,231);
18       private Vector         readerPageStart     = new Vector(50);
19       private Rectangle      readerRect          = new Rectangle();
20
21   //   variable principally for paintTextArea
22
23       private int            lowestTextAreaBaseLine = 0;
24       private int            controlSize         = 0;
25       private int            pullQuoteLineWidth   = 2;
26       private int            pullQuoteSizeIncr    = 1;
27
28       private boolean        drawPullUnderliner   = false;     //   switch to finish pull
29   quote display
30       private boolean        finishPullQuoteText  = false;     //   switch to finish pull
31   quote content
32       private boolean        lastColumn           = false;     // current setting
33       private boolean        lastRow              = false;     // current setting
34       private boolean        parSpace             = false;     // current setting
35       private boolean        priorEmptyPara       = false;     // current setting
·36
37       private Rectangle      aheadRect            = new Rectangle();
38       private Rectangle      closeRect            = new Rectangle();
39       private Rectangle      abackRect            = new Rectangle();
40
41       private Color          pullColor            = Color.blue;        // current setting
42
43       private Font           textFont             = null;      // current setting
44       private FontMetrics    textMetrics          = null;      // current setting
45       private Color          textColor            = null;      // current setting
46       private int            textHeight           = 0;         // current setting
47       private int            textAlign            = LEFT;      // current setting
48       private Rectangle      textRect             = null;      // current setting
49
50       private int            colNum = 0, colWidth = 0, colSpacing = 0, numColumns = 0,
51   oneSetOff = 0;
52       private int            nextWidth = 0, xb = 0, te = 0, xe = 0, ys = 0, yb = 0, ye = 0;
53
54       public ezrpnlap(String readerContent, ez_text ez_Text, boolean readerFrame)
55       {
56   //   dbg("ezrpnlap.<init>");
57
58           this.ez_Text = ez_Text;
59           this.readerFrame = readerFrame;
60           this.readerHeader = ez_Text.getTitle() != null;
61           this.readerContent = readerContent;
62           this.readerTextLength = readerContent.length();
63
64           if (readerFrame)
65           {
66               this.setForeground(Color.black);
67               this.setBackground(readerBackColor);
68               stepFontSize(false);
69           }
70           else
```

```
1              {
2                    this.setBackground(Color.white);      //  this is a hack
3              }
4
5              setAllFonts();
6         }
7
8         public synchronized void stepFontSize(boolean smaller)
9         {
10        //    dbg("ezrpnlap.stepFontSize(smaller = "+smaller+")");
11
12             if (smaller)      readerSizeIndex--;
13             else              readerSizeIndex++;
14
15             readerHeadIndex = readerSizeIndex + 2;
16             readerPullIndex = readerSizeIndex + 1;
17             readerFootIndex = readerSizeIndex - 2;
18
19             readerSizeIndex = Math.max(Math.min(readerSizeIndex,readerFontSize.length -
20        1),0);
21             readerHeadIndex = Math.max(Math.min(readerHeadIndex,readerFontSize.length -
22        1),0);
23             readerPullIndex = Math.max(Math.min(readerPullIndex,readerFontSize.length -
24        1),0);
25             readerFootIndex = Math.max(Math.min(readerFootIndex,readerFontSize.length -
26        1),0);
27
28             setAllFonts();
29        }
30
31        public synchronized boolean imageUpdate(Image img, int flags, int x, int y, int w,
32        int h)
33        {
34        //    dbg("ezrpnlap.imageUpdate(flags = "+flags+")");
35
36             if (formattedText && (flags & ImageObserver.ALLBITS) != 0)   //  this image is
37        complete (at last)
38             {
39        //    resetReaderView();
40                  repaintReaderScreen();
41                  return false;
42             }
43             else
44                  return true;
45        }
46
47        public final synchronized void doViewerHome()
48        {
49        //    dbg("ezrpnlap.doViewerHome()");
50
51             if (readerPageNum > 1)
52             {
53                  readerPageNum = 1;
54                  readerStartIndex =
55                       ((Integer)readerPageStart.elementAt(
56                            readerPageNum-1)).intValue();
57             }
58             repaintReaderScreen();
59        }
60
61        public final synchronized void doViewerLeft()
62        {
63        //    dbg("ezrpnlap.doViewerLeft()");
64
65             if (readerPageNum > 1)
66             {
67                  readerPageNum--;
68                  readerStartIndex =
69                       ((Integer)readerPageStart.elementAt(
70                            readerPageNum-1)).intValue();
```

```
1          }
2              repaintReaderScreen();
3          }
4
5          public final synchronized void doViewerRight()
6          {
7          //   dbg("ezrpnlap.doViewerRight(): readerEndIndex = "+readerEndIndex);
8
9              if (readerEndIndex < readerTextLength)
10             {
11                 readerPageNum++;
12                 if (readerNumPages < readerPageNum)
13                 {
14                     readerStartIndex = readerEndIndex - 1;
15                     readerPageStart.addElement(
16                         new Integer(readerStartIndex));
17                 }
18                 else    //  been there, done that
19                 {
20                     readerStartIndex =
21                         ((Integer)readerPageStart.elementAt(
22                             readerPageNum-1)).intValue();
23                 }
24             }
25             else if (readerNumPages == 0)
26             {
27                 readerNumPages = readerPageNum;
28             }
29             repaintReaderScreen();
30         }
31
32         public final synchronized void doViewerSize(boolean smaller)
33         {
34         //   dbg("ezrpnlap.doViewerSize(boolean "+smaller+")");
35
36             stepFontSize(smaller);
37             setReaderRect();
38             resetReaderView();
39             repaintReaderScreen();
40         }
41
42         public synchronized void setAllFonts()
43         {
44         //   dbg("ezrpnlap.setAllFonts()");
45
46             readerFooterFont = new Font("TimesRoman", Font.PLAIN,
47     readerFontSize[readerFootIndex]);
48             readerScreenFont = new Font("TimesRoman", Font.PLAIN,
49     readerFontSize[readerSizeIndex]);
50             readerPQuoteFont = new Font("TimesRoman", Font.PLAIN,
51     readerFontSize[readerPullIndex]);
52             readerHeaderFont = new Font("TimesRoman", Font.PLAIN,
53     readerFontSize[readerHeadIndex]);
54         }
55
56         public synchronized void setReaderRect()
57         {
58         //   dbg("ezrpnlap.setReaderRect()");
59
60             setReaderRect(this.bounds());
61         }
62
63         public synchronized void setReaderRect(Rectangle r)
64         {
65         //   dbg("ezrpnlap.setReaderRect("+r+")");
66
67             readerRect.reshape
68             (   1
69             ,   1
70             ,   r.width - 1
```

```
1                ,    r.height - 1 - (readerFrame?readerFontSize[readerFootIndex]*3/2:0)
2              );
3          }
4
5          public boolean mouseUp(Event evt, int x, int y)
6          {
7          //   dbg("ezrpnlap.mouseUp(Event "+evt+", int "+x+", int "+y+") ");
8
9              if (!readerFrame) return false;
10             else
11             if (aheadRect.inside(x,y))
12             {
13                 doViewerRight();
14                 return true;
15             }
16             else
17             if (closeRect.inside(x,y))
18             {
19                 ez_Text.disposeFrame();
20                 return true;
21             }
22             else
23             if (abackRect.inside(x,y))
24             {
25                 doViewerLeft();
26                 return true;
27             }
28             else return false;
29         }
30
31         public final Dimension minimumSize()
32         {
33         //   dbg("ezrpnlap.minimumSize()");
34
35             return new Dimension(585, 400);
36         }
37
38         public final Dimension preferredSize()
39         {
40         //   dbg("ezrpnlap.preferredSize()");
41
42             return minimumSize();
43         }
44
45         public synchronized void reshape(int x, int y, int width, int height)
46         {
47         //   dbg("ezrpnlap.reshape("+x+", "+y+", "+width+", "+height+")");
48
49             lowestTextAreaBaseLine = 0;
50             readerScreenDirty = true;
51             resetReaderView();
52
53         //   fillRect is performed here rather than in update(g)
54
55             getGraphics().setColor(getBackground());
56             getGraphics().fillRect(0, 0, width, height);
57             getGraphics().setColor(getForeground());
58
59             super.reshape(x, y, width, height);
60         }
61
62         public final synchronized void paint(Graphics g)
63         {
64         //   dbg("ezrpnlap.paint(g)");
65
66             if (readerScreenDirty)
67             {
68                 if (readerScreen != null)
69                     readerScreen.flush();
70                 readerScreen = this.createImage(this.size().width, this.size().height);
```

```
 1
 2              Graphics q = this.readerScreen.getGraphics();
 3              paintReaderScreen(q);
 4              q.dispose();
 5              readerScreenDirty = false;
 6          }
 7
 8          g.drawImage(readerScreen,0,0,this);
 9      }
10
11      private final synchronized void paintReaderScreen(Graphics g)
12      {
13      //  dbg("ezrpnlap.paintReaderScreen(g)");
14
15          Rectangle r = bounds();
16
17          setReaderRect(r);                        //  set to full area (this.bounds())
18          g.setColor(getBackground());
19          g.fillRect(0, 0, r.width, r.height);
20
21          int numCols = calculateReaderColumns();
22          if (numCols != readerNumCols)
23          {
24              readerNumCols = numCols;
25              resetReaderView();
26          }
27
28          if (readerFrame)
29              readerLeading = 2;   //  make it more comfortable
30          else
31              readerLeading = 0;   //  make it more compact
32
33          if (readerHeader && !readerFrame)    //  draw header (article title)
34          {
35              lowestTextAreaBaseLine = 0;
36              g.setColor(Color.blue);
37              int titleEnd = paintTextArea
38              (   g                       //  Graphics     g
39              ,   0                       //  int          beginIndex
40              ,   readerHeaderFont        //  Font         textFont
41              ,   ez_Text.getTitle()      //  String       textContent
42              ,   1                       //  int          numCols
43              ,   CENTER                  //  int          alignment
44              );
45
46              lowestTextAreaBaseLine += readerHeaderFont.getSize()/2;
47              readerRect.reshape
48              (   1
49              ,   lowestTextAreaBaseLine
50              ,   r.width - 1
51              ,   r.height - lowestTextAreaBaseLine
52              );
53
54              g.setColor(getBackground());
55              g.fillRect(readerRect.x, readerRect.y, readerRect.width,
56      readerRect.height);
57          }
58
59          if (demo && readerFrame)
60          {
61              Font sFont = new Font
62              (   "Dialog"
63              ,   Font.BOLD
64              ,   2*readerFontSize[readerHeadIndex]
65              );
66              setFont
67              (   Color.white
68              ,   sFont
69              ,   g
70              );
```

```
1                   String spam = "Demo";
2                   int sWide = textMetrics.stringWidth(spam), i, j;
3                   for
4                   (   j = 2*readerPerimeter + textHeight/2
5                   ;   j < readerRect.height
6                   ;   j += 3*textHeight/2
7                   )
8                       for
9                       (   i = 4*readerPerimeter + textHeight/2 - j
10                      ;   i < readerRect.width
11                      ;   i += 3*sWide/2
12                      )
13                          g.drawString     //  draw the spam
14                          (   spam
15                          ,   i + readerRect.x
16                          ,   j + readerRect.y
17                          );
18              }
19
20          g.setColor(getForeground());
21
22          readerEndIndex = paintTextArea(g, readerStartIndex);
23
24          if (readerNumPages == 0)
25          {
26              if (readerEndIndex == readerTextLength)
27                  readerEstPages = readerNumPages = readerPageNum;
28              else    //  estimate number of pages
29              {
30                  readerEstPages =
31  Math.round(0.5f+(float)readerPageNum*(float)readerTextLength/(float)readerEndIndex);
32              }
33          }
34          else
35              readerEstPages = readerNumPages;
36
37          if (readerFrame)    //  draw footer (page information)
38          {
39              StringBuffer pI = new StringBuffer();
40              pI.append("Screen ").append(String.valueOf(readerPageNum)).append(" of ");
41              if (readerNumPages == 0) pI.append("~");
42              pI.append(String.valueOf(readerEstPages));
43              String pageInfo = pI.toString();
44
45              int infoWidth = getFontMetrics(readerFooterFont).stringWidth(pageInfo);
46
47              g.setFont(readerFooterFont);
48              g.setColor(getForeground());
49              int xLoc = (r.width - infoWidth)/2;
50              int yLoc =  r.height - readerFontSize[readerFootIndex]/4 - 1;
51              g.drawString(pageInfo, xLoc, yLoc);
52          }
53      }
54
55      private final synchronized int paintTextArea
56      (   Graphics     g
57      ,   int          beginIndex
58      )                                                       //  returns string
59  index of first non-drawn character
60      {
61      //  dbg("ezrpnlap.paintTextArea(g, "+beginIndex+"): "+readerNumCols+" columns");
62
63          return paintTextArea
64          (   g
65          ,   beginIndex
66          ,   readerScreenFont
67          ,   this.readerContent
68          ,   readerNumCols
69          ,   readerAlignment
70          );
```

```
1        }
2
3        private final synchronized int paintTextArea
4        (   Graphics     g
5        ,   int          beginIndex
6        ,   Font         theFont
7        ,   String       textContent
8        ,   int          numCols
9        ,   int          alignment
10       )                                                  //   returns string
11   index of first non-drawn character
12       {
13       //   dbg("ezrpnlap.paintTextArea()");              //
14       paintTextArea(\n(g\n, "+beginIndex+"\n "+theFont+"\n, textContent\n "+numCols+"\n,
15   "+alignment+"\n)
16
17            int endText      = textContent.length();
18            int endIndex     = beginIndex;
19            boolean go = endIndex != endText;
20            numColumns = numCols;
21
22            textAlign = alignment;
23            textColor = g.getColor();
24
25            this.setFont(textColor,theFont,g);
26
27   ·        StringTokenizer paragraphs = new StringTokenizer        //   break up the text
28   into normal paragraphs
29                (textContent.substring(beginIndex),"<",true);      //   final true returns
30   token on next call
31
32            colNum = 0;
33            int colMargin = 4*textMetrics.stringWidth(BL);         //   four spaces
34   between cols
35            colWidth = (readerRect.width - 2*readerPerimeter - (numColumns-
36   1)*colMargin)/numColumns;
37            colSpacing = colWidth + colMargin;
38
39            String nextWord = MT;
40            StringBuffer thisLine = new StringBuffer(70);
41
42            xb = readerRect.x + readerPerimeter;                   //   local left edge of
43   the column area
44            te = xb;                                               //   end position of
45   this line's words
46            xe = xb + colWidth;                                    //   local right edge
47   of the column area
48
49            nextWidth = 0;
50
51            ys = readerRect.y + (textHeight + readerLeading)*3/4;  //   local starting
52   baseline of first line
53            yb = ys;
54            ye = readerRect.y + readerRect.height - textHeight/4;  //   local maximum
55   baseline of last line
56            lowestTextAreaBaseLine  = yb;                          //   lowest baseline in
57   readerRect
58            priorEmptyPara = false;                                //   permits paragraph
59   spacers
60            lastColumn = colNum == numColumns - 1;
61            lastRow = false;
62            controlSize = textHeight*2/3;
63            oneSetOff = (textHeight + readerLeading)/2;            //   use stnd font size
64   for half-space set-offs
65       //   dbg("... oneSetOff = "+oneSetOff);
66
67            abackRect.reshape
68            (   readerRect.x + readerRect.width - 2 - (controlSize+4)*3
69            ,   readerRect.y + readerRect.height - 4 - controlSize
70            ,   controlSize + 4
```

```
1                 ,    controlSize + 4
2             );
3             if (debug)
4     g.drawRect(abackRect.x,abackRect.y,abackRect.width,abackRect.height);
5
6             closeRect.reshape
7             (    readerRect.x + readerRect.width - 2 - (controlSize+4)*2
8             ,    readerRect.y + readerRect.height - 4 - controlSize
9             ,    controlSize + 4
10            ,    controlSize + 4
11            );
12            if (debug)
13    g.drawRect(closeRect.x,closeRect.y,closeRect.width,closeRect.height);
14
15            aheadRect.reshape
16            (    readerRect.x + readerRect.width - 2 - (controlSize+4)*1
17            ,    readerRect.y + readerRect.height - 4 - controlSize
18            ,    controlSize + 4
19            ,    controlSize + 4
20            );
21            if (debug)
22    g.drawRect(aheadRect.x,aheadRect.y,aheadRect.width,aheadRect.height);
23
24            draw: while (paragraphs.hasMoreTokens() && go)          //   paragraph to
25    follow
26            {
27                String para = paragraphs.nextToken("<");
28
29    //          dbg("... new paragraph token = ["+para+"]");
30
31                if (formattedText
32                    && para.startsWith("`"))                        //   test for in-line
33    title
34                {
35    //              dbg("... pull quote: yb = "+yb);
36
37                    rollBack();
38
39                    int pullQuoteHeight = 2*pullQuoteLineWidth       //   reserve pull-quote
40    bars
41                        + oneSetOff*((yb <= ys)?1:2);                //   reserve 1 or 2
42    setOffs outside bars
43
44                    textAlign = CENTER;
45                    this.setFont(Color.blue,readerPQuoteFont,g);
46
47                    pullQuoteHeight += textHeight + readerLeading;   //   reserve one
48    pullquote line of text
49    //              dbg("... pullQuoteHeight = "+pullQuoteHeight);
50
51                    if (yb + pullQuoteHeight > readerRect.y + readerRect.height)
52                    {                                                //   not enough room in
53    this (partial?) column
54                        if (lastColumn) break draw;                  //   no more columns
55    left to fill
56                        nextColumn();
57                        yb = readerRect.y;                           //   start at the top
58                    }
59                    else if (yb > ys)                                //   not at the top of
60    the column?
61                    {
62                        yb += oneSetOff;                             //   upper set-off
63                    }
64
65                    for (int line = 0; line < pullQuoteLineWidth; line++)
66                    {
67                        g.drawLine(xb, yb, xe, yb++);
68                    }
69                    yb += textHeight + readerLeading;                //   drop for next line
70                    drawPullUnderliner = finishPullQuoteText = true;
```

```
1
2                     para = para.substring(1);                    //  drop control
3  character
4                     endIndex++;                                  //  update location
5                 }
6               else if (formattedText
7                   && drawPullUnderliner && !finishPullQuoteText) //  restore stnd
8  values
9               {
10                  rollBack();
11
12                  for (int line = 0; line < pullQuoteLineWidth; line++)
13                  {
14                      g.drawLine(xb, yb, xe, yb++);
15                  }
16                  drawPullUnderliner = false;
17
18                  textAlign = alignment;
19                  this.setFont(textColor,theFont,g);
20
21                  if (yb > ys)
22                      yb += oneSetOff
23                              + textHeight + readerLeading;     //  drop for next line
24                  if (yb > ye)
25                  {                                             //  not enough room in
26  this (partial?) column
27                      if (lastColumn) break draw;               //  no more columns
28  left to fill
29                      nextColumn();
30                  }
31              }
32
33              if (para.equals("<"))                              //  an html new
34  paragraph?
35              {
36                  String tagText = paragraphs.nextToken(">").toLowerCase();
37                  int tagLength = tagText.length();
38
39                  para = paragraphs.nextToken();
40                  if (para.equals(">"))
41                  {
42                      if (tagText.startsWith("p"))
43                      {
44                          if (yb > ys && priorEmptyPara)        //  unless we're at
45  the top
46                          {
47                              yb += textHeight + readerLeading;  //  drop down to
48  another line
49                              if (yb > ye)
50                              {                                 //  not enough room in
51  this (partial?) column
52                                  if (lastColumn) break draw;   //  no more columns
53  left to fill
54                                  nextColumn();
55                              }
56                          }
57                      }
58                      lowestTextAreaBaseLine = Math.max(lowestTextAreaBaseLine,yb);
59                      priorEmptyPara = true;
60                  }
61                  endIndex += tagLength+2;
62              }
63              else if (formattedText
64                  && para.startsWith("@")
65                  && (   para.toLowerCase().endsWith(".jpeg")
66                      || para.toLowerCase().endsWith(".gif")))   //  test for in-line
67  graphic
68              {
69                  Image theImage = ez_Text.fetchImage(para.substring(1).trim());
70
```

```
1                      if (theImage != null)
2                      {
3                          rollBack();
4
5                          priorEmptyPara = false;
6
7                          int wIm = theImage.getWidth(ez_Text);
8                          int hIm = theImage.getHeight(ez_Text);
9                          if (wIm > colWidth)                          //   scale graphic down
10   to column width
11                         {
12                             hIm = Math.round( (float)colWidth * (float)hIm / (float)wIm );
13                             wIm = colWidth;
14                         }
15
16                         if (hIm > readerRect.height)                 //   scale graphic down
17   to column height
18                         {
19                             wIm = Math.round( (float)readerRect.height * (float)wIm /
20   (float)hIm );
21                             hIm = readerRect.height;
22                         }
23
24                         int fullImageHeight = hIm + ((yb > ys)?oneSetOff:0);
25                         if (yb + fullImageHeight > readerRect.y + readerRect.height)
26                         {                                            //   not enough room in
27   this (partial?) column
28                             if (lastColumn) break draw;              //   no more columns
29   left to fill
30                             nextColumn();
31                             rollBack();
32                         }
33                         else if (yb > ys)
34                         {
35                             yb += oneSetOff;                         //   upper spacing
36                         }
37
38                         int xIm = xb + (colWidth - wIm)/2;           //   center it
39
40                         g.fillRect(xIm, yb, wIm, hIm);
41                         g.drawImage(theImage, xIm, yb, wIm, hIm, this);
42                         endIndex += para.length();                   //   update location
43                         yb += hIm;                                   //   image sizing
44                         lowestTextAreaBaseLine = Math.max(lowestTextAreaBaseLine,yb);
45
46                         yb += oneSetOff
47                             + textHeight + readerLeading;            //   image spacing
48                         if (yb  > ye)                                //   no room for text
49   under image margin
50                         {
51                             if (lastColumn) break draw;              //   no more columns
52   left to fill
53                             nextColumn();
54                         }
55                     }
56                 }
57                 else                                                 //   none of the
58   above...
59                 {
60                 //   dbg("... new paragraph text:
61   "+para.substring(0,Math.min(20,para.length())));
62
63                     priorEmptyPara = false;
64
65                     if (formattedText && finishPullQuoteText)
66                     {
67                         textAlign = CENTER;
68                         this.setFont(Color.blue, readerPQuoteFont, g);
69                     }
70
```

```
1                       te = xb;                                    //   start a new line
2                       StringTokenizer words =
3                           new StringTokenizer(para,BL,true);      //   "true" returns
4   white spaces
5
6                       while (words.hasMoreTokens())
7                       {
8                           nextWord = words.nextToken();
9                           nextWidth = textMetrics.stringWidth(nextWord);
10
11                          if (te + nextWidth < xe)                //   it will fit on
12  this line, so add it on
13                          {
14                              thisLine.append(nextWord);
15                              te += nextWidth;
16                              nextWord = MT;
17                              nextWidth = 0;
18                          }
19                          else                                    //   it won't fit if
20  added, so draw the text
21                          {
22                              if (thisLine.length() > 0)          //   not if a monster
23  is first in the article
24                              {
25                          //   dbg("new line = ["+thisLine+"], nextWord =
26  ["+nextWord+"]");
27                                  endIndex += this.drawText(thisLine, g);
28                                  thisLine.setLength(0);          //   start the new line
29                              }
30
31                              if (yb > ye)                        //   start a new
32  column?
33                              {
34                                  if (lastColumn) break draw;     //   no more columns
35  left to fill
36                                  nextColumn();
37
38                                  while (thisLine.toString().startsWith(BL))
39                                  {                               // ignore preceeding
40  blanks on new line
41                                      String noBlank = thisLine.toString().substring(1);
42                                      thisLine.setLength(0);
43                                      thisLine.append(noBlank);
44                                      endIndex++;
45                                  }
46                              }
47
48                              if (thisLine.length() == 0)
49                              while (nextWord.startsWith(BL))      //   ignore preceeding
50  blanks on new line
51                              {
52                                  nextWord = nextWord.substring(1);   //   look again
53                                  nextWidth = textMetrics.stringWidth(nextWord);
54                                  endIndex++;
55                              }
56                              thisLine.append(nextWord);
57                              te = xb + nextWidth;                //   possibly a
58  monster!
59                              nextWord = MT;
60                              nextWidth = 0;
61
62                              while (te > xe)                     //   shorten the line
63  by breaking the word
64                              {
65                          //   dbg("it's big: thisLine = ["+thisLine+"]");
66                                  while (te > xe)                 //   transfer letters
67  back until fit (one line)
68                                  {
69                                      int e = thisLine.length();
```

```
1                              nextWord = thisLine.toString().substring(e-1,e) +
2    nextWord;
3                              String truncated = thisLine.toString().substring(0,e-
4    1);
5                              thisLine.setLength(0);
6                              thisLine.append(truncated);
7                         //   dbg(" - try thisLine = ["+thisLine+"]\n...and nextWord
8    = ["+nextWord+"]");
9                              te = xb +
10   textMetrics.stringWidth(thisLine.toString()+"-");
11                          }
12                          thisLine.append("-");
13                          endIndex += this.drawText(thisLine, g) - 1;
14                          thisLine.setLength(0);                    //   start the new line
15
16                          if (yb > ye)                             //   start a new
17   column?
18                          {
19                              if (lastColumn) break draw;          //   no more columns
20   left to fill
21                              nextColumn();
22                          }
23                          thisLine.append(nextWord);
24                          te = xb + textMetrics.stringWidth(thisLine.toString());
25                          nextWord = MT;
26                          nextWidth = 0;
27                      }
28                  }
29              }                                                    //   no more words in
30   this paragraph
31           if (thisLine.length() > 0)
32               endIndex += this.drawText(thisLine, g);
33           thisLine.setLength(0);                                  //   start the new line
34
35           if (yb > ye)                                            //   start a new
36   column?
37           {
38               if (lastColumn) break draw;                         //   no more columns
39   left to fill
40               nextColumn();
41           }
42           nextWord = MT;                                          //   starting fresh
43           nextWidth = 0;
44           te = xb;                                                //   pixel width of
45   this line's words
46           }
47           finishPullQuoteText = false;
48       }   //   end of draw label block                           //   no more paragraphs
49
50       while (endIndex < endText
51       && textContent.substring(endIndex++).startsWith(BL));       //   skip blanks
52
53       g.setColor(getForeground());
54   //   left arrow
55       if (readerPageNum > 1)
56       {
57           int x = abackRect.x + 2
58           ,   y = abackRect.y + abackRect.height/2;
59           for (int arrow = 0; arrow < controlSize; arrow++)
60           {
61               g.drawLine
62               (   x + arrow
63               ,   y - arrow/2
64               ,   x + arrow
65               ,   y + arrow/2
66               );
67           }
68       }
69   //   close box
70       if (readerFrame)
```

```
1                     {
2                         int x = closeRect.x + 2
3                         ,   y = closeRect.y + 2
4                         ,   w = closeRect.width  - 5
5                         ,   h = closeRect.height - 5;
6
7                         g.drawRect(x, y, w, h);
8                         g.drawLine(x, y, x+w, y+h);
9                         g.drawLine(x+w, y, x, y+h);
10                    }
11         //   right arrow
12             if (endIndex < endText)
13             {
14                 int x = aheadRect.x + aheadRect.width - 2
15                 ,   y = abackRect.y + abackRect.height/2;
16                 for (int arrow = 0; arrow < controlSize; arrow++)
17                 {
18                     g.drawLine
19                     (   x - arrow
20                     ,   y + arrow/2
21                     ,   x - arrow
22                     ,   y - arrow/2
23                     );
24                 }
25             }
26             return endIndex;
27         }
28
29         private final synchronized void resetReaderView()
30         {
31         //   dbg("ezrpnlap.resetReaderView()");
32
33             readerStartIndex     = 0;
34             readerEndIndex       = 0;
35             readerNumPages       = 0;
36             readerEstPages       = 1;
37             readerPageNum        = 1;
38             readerPageStart.removeAllElements();
39             readerPageStart.addElement(new Integer(readerStartIndex));
40         }
41
42         public final void dispose()
43         {
44         //   dbg("ezrpnlap.dispose()");
45
46             if (readerScreen != null)
47                 readerScreen.flush();
48         }
49
50         public synchronized void update(Graphics g)
51         {
52         //   dbg("ezrpnlap.update()");
53
54         //   fillRect is NOT performed here to eliminate blank screen boredom during
55     offscreen drawing
56
57         //   g.setColor(getBackground());
58         //   g.fillRect(0, 0, width, height);
59         //   g.setColor(getForeground());
60
61             paint(g);
62         }
63
64         private final synchronized void dbg(String  s)
65         {
66             if (debug) System.out.println(s);
67         }
68         private boolean debug = false;        //   Print debugging info?
69
70         private final synchronized void repaintReaderScreen()
```

```
1        {
2        //   dbg("ezrpnlap.repaintReaderScreen()");
3
4            readerScreenDirty = true;
5            repaint();
6        }
7
8        private final synchronized int calculateReaderColumns()
9        {
10       //   dbg("ezrpnlap.calculateReaderColumns()");
11
12           int numCols = 1;
13           \
14           Rectangle r = bounds(); //   rectangle of ezrpnlap
15           int numChar = 30;        //   the control parameter: min # of n's per line of
16   text
17           int targetWidth = numChar * getFontMetrics(readerScreenFont).stringWidth("n");
18
19           setReaderRect(r);
20           numCols = (r.width)/targetWidth;
21
22           if (numCols < 1)
23               numCols = 1;
24
25           return numCols;
26       }
27
28       private final synchronized int drawText(StringBuffer thisLine, Graphics g)
29       {
30       //   dbg("ezrpnlap.drawText(\""+thisLine.toString()+"\")");
31
32           g.drawString
33           (   thisLine.toString()
34           ,   xb + textAlign*(colWidth + xb - te)/2
35           ,   yb
36           );                                              //   draw the text
37           lowestTextAreaBaseLine = Math.max(lowestTextAreaBaseLine,yb);
38           yb += textHeight + readerLeading;               //   drop down to another line
39           checkShortLine();
40
41           return thisLine.length();
42       }
43
44     · private final synchronized void checkShortLine()
45       {
46       //   dbg("ezrpnlap.checkShortLine()");
47
48           if (lastColumn
49           && (lastRow = (ye - yb) <= (textHeight + readerLeading)*2))·
50           {
51               if (readerPageNum > 1)                       //   left arrow
52                   xe = abackRect.x;
53               else if (readerFrame)                        //   close box
54                   xe = closeRect.x;
55               else                                         //   right arrow
56                   xe = aheadRect.x;
57           }
58
59       //   dbg("ezrpnlap.checkShortLine(): new xe = "+xe);
60       }
61
62       private final synchronized void nextColumn()
63       {
64       //   dbg("ezrpnlap.nextColumn()");
65
66           colNum++;                                        //   finished another column
67           lastColumn = colNum == numColumns - 1;
68       //  ·clearInitialSpaces = true;
69
```

```
1          yb = ys;                                    //  local baseline of first
2   line
3          xb += colSpacing;                           //  local left edge of the
4   column area
5          te += colSpacing;                           //  local end of text
6          xe += colSpacing;                           //  local right edge of the
7   column area
8
9          checkShortLine();
10      }
11
12      private final synchronized void setFont(Font f, Graphics g)
13      {
14      //  dbg("ezrpnlap.setFont("+f+")");
15
16          textMetrics = getFontMetrics(f);
17          textHeight = f.getSize()*5/4;
18          g.setFont(f);
19
20      //  dbg("ezrpnlap.setFont(setFont): textHeight = "+textHeight);
21      }
22
23      private final synchronized void setFont(Color c, Font f, Graphics g)
24      {
25      //  dbg("ezrpnlap.("+c+", "+f+", g)");
26
27          g.setColor(c);
28          setFont(f, g);
29      }
30
31      private final synchronized void rollBack()
32      {
33      //  dbg("ezrpnlap.rollBack()");
34
35          if (yb == ys && drawPullUnderliner && !finishPullQuoteText)
36          {
37              yb = readerRect.y
38                      + readerRect.height
39                      - pullQuoteLineWidth;               //  roll back to base of
40   previous column
41              colNum--;
42              lastColumn = false;
43
44              xb -= colSpacing;                           //  local left edge of the
45   column area
46              te -= colSpacing;                           //  local end of text
47              xe -= colSpacing;                           //  local right edge of the
48   column area
49          }
50          else if (yb > ys)
51          {
52              yb -= (textHeight + readerLeading)/2;       //  roll back half of last CR
53          }
54          else
55          {
56              yb = readerRect.y;                          //  start at the top
57          }
58      }
59  }
60
61  /**
62   *  sample web page with applet tag
63   */
64  <HTML>
65  <HEAD>
66    <META NAME="GENERATOR" CONTENT="Adobe PageMill 2.0 Mac">
67    <TITLE></TITLE>
68  </HEAD>
69  <BODY>
70
```

```
1    <applet codebase="ez_class" code="ez_text.class" width=300 height=200>
2    <param name="fileName" value="ezsample.txt">
3    </applet>
4
5    </BODY>
6    </HTML>
7
8    /**
9     *  ezsample.txt web page for applet
10    */
11   `EZ Text Sample Article
12   `"Click on the applet"
13   The reader screen presentation of content is interactive and optimized for
14   readability. Font size can be changed by pressing up or down arrow keys, or by
15   clicking the Font+ or Font- buttons. Pages can be turned by pressing home, left, or
16   right keys, or by clicking the Home, Previous, or Next buttons.
17   `"optimized for readability"
18   One of the key advantages of this reader screen presentation is its careful avoidance
19   of a particular form of eye-strain known as optokinetic nystagmus. Nystagmus, a
20   reflexive jump in the muscles directing the eye, is triggered by, for example,
21   vertical scrolling of text on a computer screen.
22   @ez_eye.gif
23   As nystagmus strain increases, muscle jumps can become exaggerated and even erratic,
24   finally resulting in a perception of visual weariness. Nystagmus is well-known as a
25   side-effect of excessive alcohol: a simple road-side test conducted by law officers
26   can aid in determination of a possible DWI offense. Other possible causes of nystagmus
27   include general physical exhaustion and illness.
28
29   While the term "nystagmus" may have been unfamiliar, the condition is encountered in
30   many circumstances, none of which is very pleasant.
31   `"optokinetic nystagmus"
32   The main side effect of optokinetic nystagmus induced by scrolling text is a tendency
33   to print the document, so that it can be read later with greater comfort. This
34   interrupts delivery of content and consumes resources. It is very possible that the
35   demise of the paperless office dreams of the '80's was caused in part by nystagmus.
36   Another major cause of its demise was the difference in formatting between standard
37   paper, height being typically 30% greater than width, and standard computer monitors,
38   width being typically 33% greater than height. Word processors targeted paper
39   formatting, so that documents delivered from computer to computer were not easily
40   adapted to screen formats. This pattern also resulted in a tendency to print such
41   documents rather than struggle to consume their contents on screen.
42
43   EZ Text solves these twin problems nicely. First, there is no scrolling text,
44   resulting in reduced eye-strain. Font sizing can be adjusted by the reader for his
45   particular situation of lighting, monitor quality and size, and his personal visual
46   comfort. Second, The content is automatically laid out in columns of text, with
47   neither too few words in a line, about 30 characters minimum, nor too many, about 60
48   characters maximum. The upper limit yields a line length which, as the human eye
49   executes its rapid raster motion to the beginning of the next line, is not too long to
50   accurately and easily find the correct next new line. If lines are much shorter, below
51   the minimum, lines of text contain too little content, and the frequent raster motions
52   can induce eye-strain.
53   `"Web pages ... become a rewarding experience with EZ Text."
54   This sample document contains about 3300 characters in about 640 words. At normal
55   reading speeds, it should take about 5 minutes to absorb its main content. The effort
56   required to read this article on the screen, using EZ Text, should be noticeably less
57   than required by other channels. For instance, using a word processor, two approaches
58   are common. The first is to print it out, collect the page(s), and finally read it.
59   Alternatively, one could open the document in a word processor, adjust window widths
60   and zoom factors, and finally read it, occasionally scrolling to bring unseen parts
61   into view. With EZ Text, just sit back, tap the up arrow to adjust the font, and tap
62   the right arrow to turn the page as needed. Web pages, known widely as challenging
63   when it comes to delivering on-screen content, can become a rewarding experience with
64   EZ Text.
65
```